OPERATIONS AUTOMATION

Charles Thomas Boreham
OAO Corporation
787 West Woodbury Road
Altadena, CA 91001

ABSTRACT

This is truly the era of "Faster-Better-Cheaper" at the National Aeronautics and Space Administration/Jet Propulsion Laboratory (NASA/JPL). To continue JPL's primary mission of building and operating interplanetary spacecraft, all possible avenues are being explored in the search for better value for each dollar spent. A significant cost factor in any mission is the amount of manpower required to receive, decode, decommutate, and distribute spacecraft engineering and experiment data. The replacement of the many mission-unique data systems with the single Advanced Multimission Operations System (AMMOS) has already allowed for some manpower reduction. Now, we find that further economies are made possible by drastically reducing the number of human interventions required to perform the setup, data safing, station handover, processed data loading, and tear down activities that are associated with each spacecraft tracking pass.

We have recently adapted three public domain tools to the AMMOS system which allow common elements to be scheduled and initialized without the normal human intervention. This is accomplished with a stored weekly event schedule. The manual entries and specialized scripts which had to be provided just prior to and during a pass are now triggered by the schedule to perform the functions unique to the upcoming pass.

This combination of public domain software and the AMMOS system has been run in parallel with the flight operation in an online testing phase for six months. With this methodology, a savings of 11 man-years per year is projected with no increase in data loss or project risk. There are even greater savings to be gained as we learn other uses for this configuration.

INTRODUCTION

The purpose of this paper is to explain what has been done to automate the operation of the Multimission Ground Data System (MGDS) at JPL. It is the further intent of this paper to explain some of the problems encountered during the systems' evolution that prevented this automation from occurring earlier.

OBJECTIVES

The implementation of JPL's automation of MGDS operations addressed seven objectives:

[1] Automate the operation of telemetry processing for realtime operations thus eliminating all of the repeated tasks that the system controller would normally perform manually during a support period.

[2] Accomplish the automation in a simple yet reliable fashion.

[3] Maintain the ability for system controller intervention.

[4] Provide automatic backup to MGDS systems in case of hardware or operating system failure.

[5] Use a method independent of applications software and hardware platforms.

[6] Eliminate labor-intensive operational work-arounds associated with unstable or incomplete applications software deliveries.

[7] Give the system the flexibility to easily accommodate additional functions and/or projects.

PROBLEMS

Providers of realtime support are always interested in minimizing costs and maximizing reliability through the automation of operator tools. A series of obstacles have persisted, however, that have held back the automation process.

[1] The cost of operations is higher than necessary because systems are frequently delivered strictly to meet budget and schedule constraints. Such a delivery is made with the absolute minimum capabilities that will meet project processing requirements, and no emphasis is placed on operability issues. Yet, the prime focus of operations groups is not that deliveries meet specific requirements. Rather, it is that deliveries produce the data products required by projects without extensive human intervention. So when deliveries are rushed in order to meet budget and schedule constraints, they lack operability and the operational costs are increased.

Further, the automation of a system is not an achievable goal if the hardware and software are not stable. Thus, the significant reduction in costs available from the automation of operations also hinges upon the operability of delivered systems.

[2] For a variety of reasons, there is strong pressure to adopt a Graphical User Interface (GUI) strategy for all levels of applications. This type of interface is beneficial for occasional users of the system, but not for operations personnel who maintain and run the system around the clock and who understand the system's full capabilities. Operations personnel need to be able to act quickly at all levels of each application and its operating system. From an operational perspective, a punch and click type of interface is intrusive, limiting, and cumbersome, and is thus an obstacle to any type of automation that would lower operational costs.

[3] The concept of running a system from a Sequence of Events (SOE) file with little or no human intervention is not a new one. But the implementation of this concept, too, has had its associated problems. One of these is the high frequency of changes that are applied to any given weekly SOE. Historically, these changes have had to be applied manually, forcing frequent operator intervention.

Thus, the question to be answered became: *Could the operation of the MGDS system be automated to the degree that we desired using available software and with the system design that was already online?* With a little creativity and a thorough understanding of the operational functions, this goal turned out to be achievable.

The UNIX utilities that are being applied in JPL's automation are straight forward and available to all users. The third party software programs are available on the network and once again can be accessed and used by anyone. Not only did we accomplish the objectives set out earlier; the implementation of these automated operations features resulted in an operational staffing reduction from 28 to 17 for the same data delivery workload. On an annual basis this saves JPL approximately 1.2 million dollars in operational costs.

THE ROLE OF UNIX

When considering the automation of realtime operations, we frequently tend to see large complicated software programs that cost as much as the current operators who run the systems. With the operating systems that were previously in use, this assessment would have been accurate. But with the adoption of UNIX as the operating system and with the tools and utilities that then become available, the cost of automation is within the reach of all groups.

JPL commenced its transition to the UNIX Operating System in 1986. The first version of the flight applications that ran in the UNIX environment was V7, which supported the Magellan mission, but required extensive operational work-arounds. V7 had to be monitored continuously by the realtime operations group to ensure the delivery of usable data to the project. As previously described, the stability of the realtime applications software is a key factor in successfully automating operational tasks. In our case, this needed stability was achieved in December, 1993, with the delivery of the nineteenth major version of the application software. This delivery allowed our operations staff to take advantage of the tools, utilities, and public domain software packages that are available for UNIX.

Using off-the-shelf and public domain software with a small amount of custom coding, we were not only able to achieve a high degree of autonomous operation but also to build an inexpensive, software-switched, fault-tolerant system. We never lose data due to a host system failure. This general approach can be applied to a broad variety of high reliability applications at a fraction of the cost of the special purpose fault-tolerant computing systems on the commercial market. Moreover, this solution is vendor platform independent, requiring only a UNIX operating system environment.

THE ROAD TO AUTOMATION

The reliability of delivered applications paved the way for automated control. The operations task for flight projects is repetitive and can therefore be scripted to run on a schedule. This was done on our systems by combining a seven day SOE, custom software to convert the schedule to applications directives, public domain software, and UNIX utilities.

The integration of COTS and public domain software into realtime mission-critical systems is a viable and cost-effective alternative to custom designed and developed code. The automated operational capability described in this paper was conceived and integrated in a two month period by selected individuals in the operations group as time permitted. Parallel testing took an additional six months. *Under the automated configuration, more spacecraft data arrived at the projects' databases than under the manual system*!

WHAT IS AUTOMATED?

We maintain at least 32 applications and 10 monitoring processes on 35 remotely accessed systems. Prior to automating operations this same configuration was maintained manually. In the following paragraphs we describe details of what is currently automated in our implementation. In addition, we describe some of the specific components that we used.

At the heart of the configuration is the seven-day SOE. From this, all associated jobs are derived and submitted to the system for the full week for all monitored spacecraft. Such job schedules are disk based in UNIX and therefore remain scheduled even when the host system is brought back from a failure. This means that all scheduled jobs will still execute when the system is brought back to online status. Jobs that did not execute when the host was down have to be entered manually but all jobs are scripted and well-ordered, and can thus be resubmitted to the system easily.

screen

In the UNIX world, software follows a standard input/output protocol that previously created a major problem for application and system failure recovery. If a host system failed, the applications that were being run from that host by remote login also failed. This difficulty was resolved by utilizing **screen**, a public domain software program written by Oliver Laumann of the Technical University of Berlin. Here, predefined scripts start **screen** prior to starting the applications. Standard input and output are buffered by **screen** on the X terminals

harboring remote logins, so that when the host system fails, the applications continue to run. A mechanism for reattaching to the application is also provided by **screen** so that operations can be normalized once the host is back on line. Now, when the host system goes down, there is no data loss during the host's down period. All remote systems continue the processing and loading of project data during a host failure.

System Utilities

To recover from failures of the host system we have used a number of UNIX capabilities: First we have the failed host automatically reboot itself. Next, we provide that X Windows accesses a customized initialization when the host comes up. The initialization file creates all appropriate windows and remote logins that were being used prior to the failure. The host then accesses a script that reattaches its windows to the proper processes (using **screen**) which have remained unaffected despite the failure of the host. Again, downstream users of the data will not have been affected by the failure of the host system.

monitor

The system is also protected against a total hardware failure of the host system. The operations group has built a program that runs on the backup system and monitors the prime host. If a failure of the prime host occurs, a five minute timer is set on the monitoring system, and a popup window notifies the controller immediately that the prime host has failed. The controller can respond to the popup window informing the backup system to promptly usurp the duties of the prime host. If, on the other hand, the popup is not responded to within the five minutes, the backup system automatically executes the X Windows initialization file and reattaches to the appropriate processes using **screen**. The backup host thus assumes all control and processing for the prime host. Once again, the downstream user of the data is not affected.

The problem of applications failures on the remote systems is handled by additional monitors. If an application or its remote system fails, a popup window notifies the controller so that the hardware can be substituted or the software problem can be properly handled. The popup window is activated frequently and has a very annoying beep that cannot be ignored.

expect

We use a public domain package called **expect** written by Don Libes of the National Institute of Standards and Technology. This utility is set up to acquire and update a copy of the seven-day schedule. The output of **expect** (the seven-day schedule) is piped through another piece of custom software written by the operations group. That output is a file of the scripting schedule that is to be submitted to the system. Scripts are scheduled using the UNIX utility, **at**. With **expect**, updates are made to the schedule automatically without the need for human intervention.

force

The third public domain package used is **force**, which was written by Jeff Glass of the MITRE Corporation. We use this essential utility to place the applications-level commands on their associated windows. The applications commands and their **force** directives reside in the predefined scripts that are submitted by **at** to execute at specific times according to the seven-day schedule. We have also modified the **xterm** program and updated the UNIX device directory to incorporate the use of special ttys. We dedicated these ttys to each of the X terminals being used so that the

same tty names always assign to the same windows. This was important because, while the command that **force** sends is guaranteed by TCP/IP to reach its destination, **force** knows nothing of the context of that destination. Scripts can now consistently **force** commands to given windows with confidence that the assumed context is valid.

CONCLUSION

The automation of the operation of our system has been accomplished with some very simple concepts and tools, using scripts, minor amounts of C programming, and public domain software. There was no significant expense involved, and the outcome has been the dramatic reduction of 11 man-years per year in the cost of operations.